Phase -3

Code description

PUBLIC HEALTH AWARENESS

Team members

Sasikala

Arathi

Savitha

Sudhersan reddy

Bhanu Prakash

Importing library

Import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

print(‘Successfully imported’)

Importing dataset

Data = pd.read_csv('/kaggle/input/mental-health-in-tech-survey/survey.csv')

data.head()

Preprocessing and cleaning dataset

Check the data set for missing data

```
If data.isnull().sum().sum() == 0 :
    print ('There is no missing data in our dataset')
else:
    print('There is {} missing data in our dataset
'.format(data.isnull().sum().sum()))
```

Check our missing data from which volume and how many unique features they have

Frame = pd.concat([data.isnull().sum(), data.nunique(), data.dtypes], axis = 1, sort= False)

frame

Look at what is in the ‘Work_interfere’ column to choose a suitable method to fill nan values.

Data[‘work_interfere’].unique()

Plot **work_interfere**
Add the value of each parametr on the Plot

```
ax = sns.countplot(data  = data , x = 'work_interfere');
ax.bar_label(ax.containers[0]);
```

```
From sklearn.impute import SimpleImputer
import numpy as np
columns_to_drop = ['state', 'comments', 'Timestamp']
for column in columns_to_drop:
    if column in data.columns:
        data = data.drop(columns=[column])
```

Fill in missing values in work_interfere column

```
data['work_interfere'] = np.ravel(SimpleImputer(strategy = 'most_frequent').fit_transform(data['work_interfere'].values.reshape(-1,1)))
data['self_employed'] = np.ravel(SimpleImputer(strategy = 'most_frequent').fit_transform(data['self_employed'].values.reshape(-1,1)))

data.head()
```

Bar chart representation

```
Ax = sns.countplot(data=data, x='work_interfere');
ax.bar_label(ax.containers[0]);
```

Check unique data in gender columns

print(data[‘Gender’].unique())

print(‘’)

print(‘-’*75)

print(‘’)

Check number of unique data too.

Print(‘number of unique Gender in our dataset is :’, data[‘Gender’].nunique())

Gender data contains dictation problems, nonsense answers, and too unique Genders.

```
data[‘Gender’].replace([‘Male ‘, ‘male’, ‘M’, ‘m’, ‘Male’, ‘Cis Male’,
          ‘Man’, ‘cis male’, ‘Mail’, ‘Male-ish’, ‘Male (CIS)’,
          ‘Cis Man’, ‘msle’, ‘Malr’, ‘Mal’, ‘maile’, ‘Make’,], ‘Male’, inplace = True)

data[‘Gender’].replace([‘Female ‘, ‘female’, ‘F’, ‘f’, ‘Woman’, ‘Female’,
          ‘femail’, ‘Cis Female’, ‘cis-female/femme’, ‘Femake’, ‘Female (cis)’,
          ‘woman’,], ‘Female’, inplace = True)

data[“Gender”].replace([‘Female (trans)’, ‘queer/she/they’, ‘non-binary’,
          ‘fluid’, ‘queer’, ‘Androgyne’, ‘Trans-female’, ‘male leaning androgynous’,
          ‘Agender’, ‘A little about you’, ‘Nah’, ‘All’,
          ‘ostensibly male, unsure what that really means’,
          ‘Genderqueer’, ‘Enby’, ‘p’, ‘Neuter’, ‘something kinda male?’,
          ‘Guy (-ish) ^_^’, ‘Trans woman’,], ‘Other’, inplace = True)

print(data[‘Gender’].unique())
```

Plot Genders column after cleaning and new categorizing

```
ax = sns.countplot(data=data, x='Gender');
ax.bar_label(ax.containers[0]);
```

Our data is clean now ? Let’s see.

```
If data.isnull().sum().sum() == 0:
    print('There is no missing data')
else:
    print('There is {} missing data'.format(data.isnull().sum().sum()))
```

Lt’s check duplicated data.

```
If data.duplicated().sum() == 0:
    print(‘There is no duplicated data:’)
else:
    print(‘Tehre is {} duplicated data:’.format(data.duplicated().sum()))
    #If there is duplicated data drop it.
    Data.drop_duplicates(inplace=True)

print(‘-’*50)
print(data.duplicated().sum())
```

Look unique data in Age column

data[‘Age’].unique()

We had a lot of nonsense answers in the Age column too.

This filtering will drop entries exceeding 100 years and those indicating negative values.

```
Data.drop(data[data[‘Age’]<0].index, inplace = True)

data.drop(data[data[‘Age’]>99].index, inplace = True)


print(data[‘Age’].unique())
```

Let’s see the Age distribution in this dataset.

```
Plt.figure(figsize = (10,6))
age_range_plot = sns.countplot(data = data, x = ‘Age’);
age_range_plot.bar_label(age_range_plot.containers[0]);
plt.xticks(rotation=90);
```

In this plot moreover on Age distribution we can see treatment distribution by age

plt.figure(figsize=(10, 6));

sns.displot(data[‘Age’], kde = ‘treatment’);

plt.title(‘Distribution treatment by age’);

- #Check Dtypes data.info()

Use LabelEncoder to change the Dtypes to ‘int

’

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

#Make the dataset include all the columns we need to change their dtypes

columns_to_encode = [‘Gender’, ‘Country’, ‘self_employed’,’family_history’, ‘treatment’, ‘work_interfere’,’no_employees’,
‘remote_work’, ‘tech_company’,’benefits’,’care_options’, ‘wellness_program’,
‘seek_help’, ‘anonymity’, ‘leave’, ‘mental_health_consequence’, ‘phys_health_consequence’,
‘coworkers’, ‘supervisor’, ‘mental_health_interview’,’phys_health_interview’,
‘mental_vs_physical’, ‘obs_consequence’]

#Write a Loop for fitting LabelEncoder on columns_to_encode

for columns in columns_to_encode:
    data[columns] = le.fit_transform(data[columns])

data.info()

- #Let’s check Standard deviation
- data.describe()

```
From sklearn.preprocessing import MaxAbsScaler, StandardScaler

data[‘Age’] = MaxAbsScaler().fit_transform(data[[‘Age’]])
data[‘Country’] = StandardScaler().fit_transform(data[[‘Country’]])
data[‘work_interfere’] =
StandardScaler().fit_transform(data[[‘work_interfere’]])
data[‘no_employees’] =
StandardScaler().fit_transform(data[[‘no_employees’]])
data[‘leave’] = StandardScaler().fit_transform(data[[‘leave’]])

data.describe()
```

Split the data to train and test

```
From sklearn.model_selection  import train_test_split

#I wanna work on ‘treatment’ column.
X = data.drop(columns = [‘treatment’])
y = data[‘treatment’]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

print(X_train.shape, y_train.shape)
print(‘-’*30)
print(X_test.shape, y_test.shape)
print(‘_’*30)
```

```
From sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA
from sklearn.tree import DecisionTreeClassifier as DT
```

Random forest classifier

```
Steps_rfc = [('Scaler', StandardScaler()),
          ('clf', RFC(n_estimators = 40))]

clf_rfc = Pipeline(steps=steps_rfc)

clf_rfc.fit(X_train, y_train)

y_pred_rfc = clf_rfc.predict(X_test)
print('RFC accuracy: ', accuracy_score(y_true=y_test, y_pred=y_pred_rfc)*100)
```

K nearest neighbor

```
Steps_knn = [('Scaler', StandardScaler()),
          ('clf', KNN(n_neighbors = 5))]
```

```
clf_knn = Pipeline(steps=steps_knn)
```

```
clf_knn.fit(X_train, y_train)
```

```
y_pred_knn = clf_knn.predict(X_test)
print('KNN accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_knn)*100)
```

Support vector classifier

```
Steps_svc = [('Scaler', StandardScaler()),
             ('clf', SVC())]



clf_svc = Pipeline(steps=steps_svc)


clf_svc.fit(X_train, y_train)

y_pred_svc = clf_svc.predict(X_test)
print('SVC accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_svc)*100)
```

Decision tree

```
Steps_dt = [('Scaler', StandardScaler()),
            ('clf', DT())]


clf_dt = Pipeline(steps=steps_dt)


clf_dt.fit(X_train,  y_train)

impeded = clf_dt.predict(X_test)
print('DT accuracy :', accuracy_score(y_true=y_test,  y_pred=y_pred_dt)*100)
```